

Architecting the Human Space Flight Program with Systems Modeling Language (SysML)

Maddalena M. Jackson¹, Michela Muñoz Fernández², Thomas I. McVittie³, and Oleg V. Sindiy⁴
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California, 91109

The next generation of missions in NASA's Human Space Flight program focuses on the development and deployment of highly complex systems (e.g., Orion Multi-Purpose Crew Vehicle, Space Launch System, 21st Century Ground System) that will enable astronauts to venture beyond low Earth orbit and explore the moon, near-Earth asteroids, and beyond. Architecting these highly complex system-of-systems requires formal systems engineering techniques for managing the evolution of the technical features in the information exchange domain (e.g., data exchanges, communication networks, ground software) and also, formal correlation of the technical architecture to stakeholders' programmatic concerns (e.g., budget, schedule, risk) and design development (e.g., assumptions, constraints, trades, tracking of unknowns).

This paper will describe how the authors have applied System Modeling Language (SysML) to implement model-based systems engineering for managing the description of the End-to-End Information System (EEIS) architecture and associated development activities and ultimately enables stakeholders to understand, reason, and answer questions about the EEIS under design for proposed lunar Exploration Missions 1 and 2 (EM-1 and EM-2).

I. Introduction

This paper describes the use of Model-Based Systems Engineering (MBSE) with SysML to enable more robust and complete systems engineering and integrated analysis of complex System-of-Systems (SoS) problems which have historically been implemented via paper/presentation-based design capture, disparate models, in documents, and in the brains of expert engineers across many disciplines. This paper describes constructs, rules, and methods applied in developing a model conforming to a rigorous and engineered structure that supports views for domain experts, analysis and tracking of stakeholder concerns, and standardized analysis that uses the relationships between the traditional system views to provide cross-cutting reasoning.

A. Background - To the Moon with Exploration Missions 1 and 2

Exploration Mission 1 (EM-1) will send an unmanned Orion Multi-Purpose-Crew-Vehicle (MPCV) around the moon on a 7-to-10 day mission to qualify NASA's new Space Launch System (SLS), and verify Orion MPCV's readiness to carry astronauts beyond Earth orbit. EM-1 is targeted for launch in late-2017. Subsequently, Exploration Mission 2 (EM-2) will send a crew of four astronauts on a 10-to-14 day mission to the moon, where astronauts will spend four days in lunar orbit to test critical mission events and perform operations in relevant environments to support exploration to more distant points beyond Earth orbit. EM-2 is targeted for launch in mid-2019.

To accomplish these missions we must coordinate a significant number of diverse terrestrial and space-based systems including: MPCV; SLS; supporting space and terrestrial communication facilities (provided the USAF,

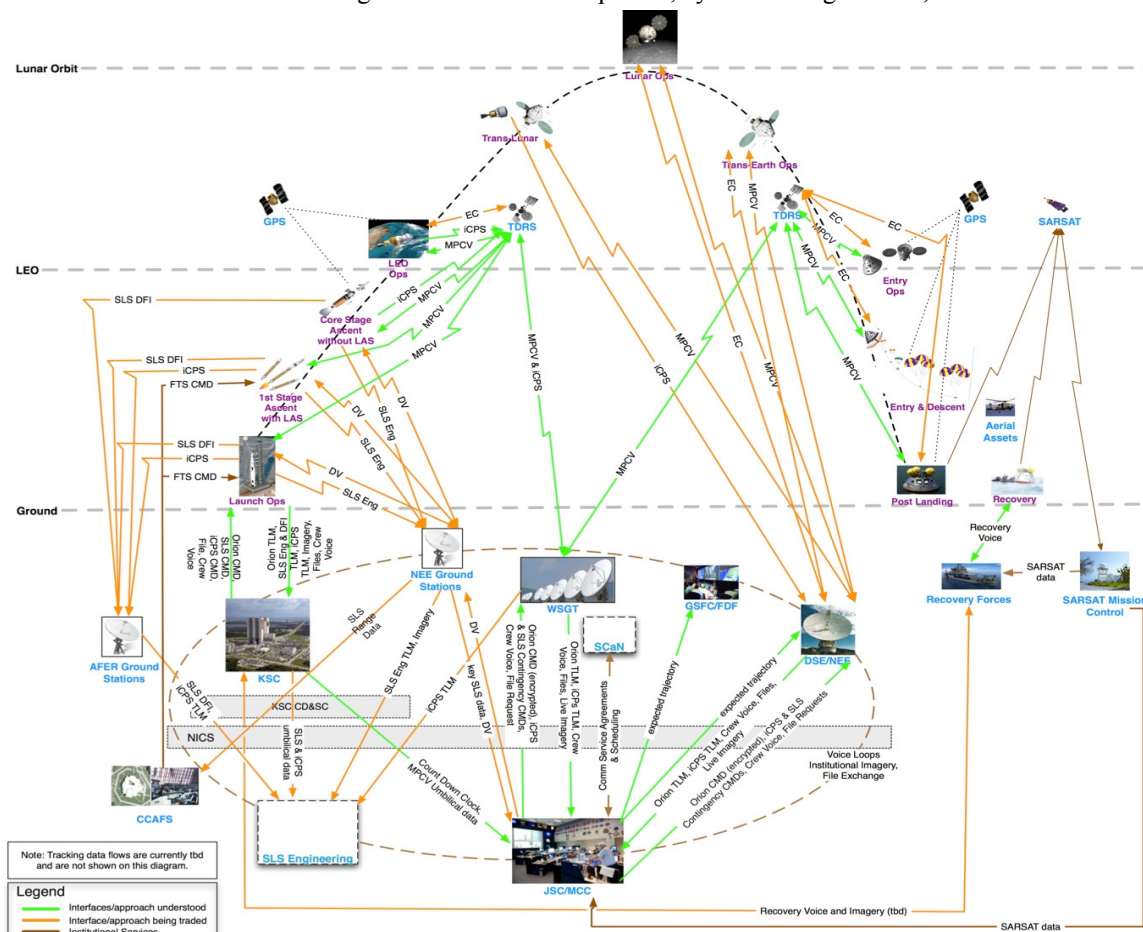
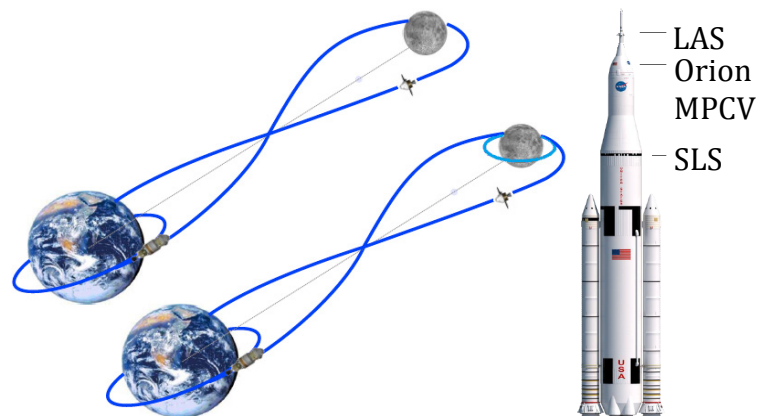
¹ Software Systems Engineer, Ground System Architecture and Systems Engineering, 4800 Oak Grove Dr., M/S 301-285, Pasadena, CA 91109; AIAA Member.

² Systems Architect, Ground System Architecture and Systems Engineering, 4800 Oak Grove Dr., M/S 264-767, Pasadena, CA 91109; AIAA Member.

³ Principle Software Systems Engineer, Ground System Architecture and Systems Engineering, 4800 Oak Grove Dr., M/S 301-480, Pasadena, CA 91109; AIAA Member.

⁴ Systems Architect, Ground System Architecture and Systems Engineering, 4800 Oak Grove Dr., M/S 301-285, Pasadena, CA 91109; AIAA Member.

These systems are in turn comprised of hardware and software subsystems connected to each other via a variety of hardline and RF communications links that support the exchange of critical data such as Commands (CMD), various forms of Telemetry (e.g., Operational, Developmental, Engineering), File Exchanges, Primary and Dissimilar Voi



0

and utilized communications links.

End-to-End Information Systems (EEIS) System Engineering refers to the process that Systems Engineers use to capture the mission's configuration and assess its ability to meet the mission's needs. It requires interaction with project management and technical stakeholders to define their concerns (e.g., needs, requirements, constraints), and capturing/integrating the work of multiple technical subject matter experts (e.g., communications engineers, missions operations engineers, avionics engineers, etc.) into a cohesive set of integrated products that address those concerns (in IEEE 1471 terminology these products are called "viewpoints").

Since the stakeholders of the EM-1 and EM-2 missions share many of the same concerns as their counterparts in the NASA's earlier Constellation Program (CxP)¹ and the Exploration Flight Test (EFT-1) mission^{2,3}, we chose to reuse the EEIS viewpoints that they had defined.

Figure 3 provides an overview of the viewpoints defined for the CxP- and EFT-1 missions including the primary set of questions (i.e., concerns) that each viewpoint addresses.

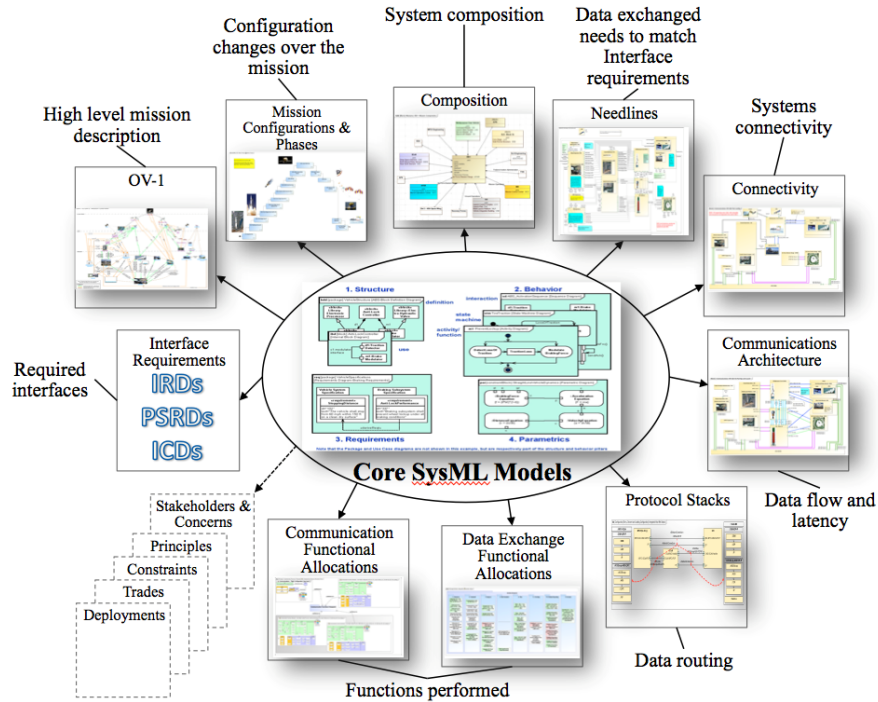


Figure 3: EEIS Architecture Description Viewpoints

The viewpoints and their associated views are, as the figure indicates, all produced from the same core model. The core model is made up of structures, relationships, and attributes that capture and address the union of all stakeholder concerns. Each view is a projection of the core model onto a diagram, where the elements on the diagram address the specific subset of concerns of the stakeholder(s) using the specific set views.

Our work on adding robustness and rigor to the core SysML Models is (initially) scoped to a subset of these viewpoints, key aspects of which are described below:

1. Interface Requirements

Define the high-level requirements that govern the connections (e.g., network communications, exchanges of data, etc.) between systems. The viewpoint captures not only the relationship between requirements (e.g., a producer/consumer relationship), but it also captures the requirement's approval, funding, and implementation status.

2. Mission Configurations and Phases Viewpoint:

The views in this viewpoint address the need to define and communicate the major mission phases (including both operations and test/integration), their ordering, and the transitions between them. Examples of mission phases would include: Launch Operations, 1st stage ascent with LAS, and Low Earth Orbit Operations.

3. *Needlines Viewpoint:*

The needlines viewpoint defines the required exchanges of information between systems during each mission phase. The Needlines Viewpoint derives from the DoDAF Operational View-2⁴, in which a needline represents exchanges of resources, existing or required. Needlines contain common attributes such as mission phase, mission importance, source and sink of the data exchange, maximum allowed latency, completeness, worst case data rate, expected data rates, and requirements related to integrity and confidentiality.

4. *Connectivity Viewpoint:*

The connectivity viewpoint defines how the systems are connected using various physical communications media (such as RF, Wide-Area-Network, umbilical, etc.) and systems that make up the networks (e.g., network routers, switches, firewalls, distribution gateways, etc.). The views for this viewpoint sometimes show facilities within larger organizations and sites to provide ownership and location clarity. In addition to common attributes such as availability and bandwidth, key attributes of a communications link may change depending on the medium: for example, RF connections will track the frequency, modulation, bit error rates, etc., while physical connections will track number of circuits, their data rates, and maximum drive lengths.

5. *Protocol Stacks Viewpoint:*

This viewpoint depicts the protocol stacks that are built, exchanged, and extracted by systems involved in each data exchange or software/communications link. The protocol stacks viewpoint modified for EFT-1² presents the stacks using a style advocated in CCSDS's Reference Architecture for Space Data Systems (RASDS) Recommended Practice document⁵ and refines the example viewpoint provided therein.

B. Learning from our EEIS modeling for CxP and EFT-1

In both the CxP and EFT-1 EEIS architecture definition activities the systems engineering and integration teams used models to generate views from predefined set of governing viewpoints. These viewpoints were successful in effectively communicating the configuration of the systems and were widely used to communicate the evolving architecture description to both program management and technical staff. The models themselves also provided several advantages: (1) a natural point where the systems engineering team could collect key data that had previously been scattered across the program and maintained in a variety of documents and presentations, (2) the contents of the various viewpoints were always consistent with each other since they were derived from a common 'single-source-of-truth' model, and (3) the process of defining the models forced the technical teams to be more rigorous in defining the involved systems, interfaces, and requirements.

Prior to beginning the EM-1 and EM-2 EEIS architecture definition efforts, we reviewed the prior CxP and EFT-1 models and identified areas where we should strive to improve. These fell into four major categories:

- 1) *Better represent the impact of mission phases on the system configurations and views.* In our prior models, the relationship between a mission phase and a view was largely represented by appending the name of the phase to the names of the views, data exchanges, and links. Similar name-based approaches were used to identify software and communications links that were likewise associated with a phase. While workable, this made it extremely cumbersome to query the model and extract the configurations associated with a specific phase.
- 2) *More powerful linkages between the various levels of the EEIS technical architecture and the viewpoints.* In our prior models, the various EEIS technical viewpoints (e.g., data exchanges, software, networks, communications, etc.) were very loosely tied together. While a reasonably informed reader could look at the various viewpoints and intuit the connections between the various views, the connections were not really represented in the underlying models (and thus inaccessible to model queries, and formal reasoning and analysis).
- 3) *More formally represent data as attributes of the model.* In previous efforts the systems engineering team concentrated on capturing data—often as text labels or tags. Unfortunately this was rather unstructured which made it difficult to retrieve and use the data in analyses.
- 4) *Provide the ability to ask the model more sophisticated questions.* In previous models, we were able to ask the model basic questions, but the lack of the capabilities noted above prevented us from being able to address more complex stakeholder concerns such as:
 - a. *Can the system downlink all of the critical data, within the latency requirements, over the communications links available during this phase?* Addressing this question requires us to be able to not only understand the communication and data exchange configurations during this phase, but also the data rates, and overhead added to the exchanges by the various protocol stacks. While a

good portion of this data was in the model, it was not in a form that would allow us to form this query.

- b. *Are there phases when Mission Systems will not be able to communicate with the crew on MPCV?* This involves not only knowledge of the communications links, but also the precise configuration of the vehicles during that phase (i.e., where are the antennas and which are occluded?).
- c. *What is the impact on our ability to operate the mission if we encounter a cyber-attack on a particular communications link or system?* Requires us to be able to reason about the key functions that are on each system; how those systems are connected via software interchanges, networks and communications links; and potentially even how firewalls constrain how the networks behave.

To formulate the EM-1/EM-2 EEIS architecture description modeling approach, we assembled a team comprised of EEIS subject matter experts and SysML modelers. The consensus of the team was that our approach should retain the foundational viewpoints that were defined for CxP and EFT-1, but that they should be based on a rigorous framework comprised of meta-models, profiles, and patterns. Further, that the models should classify the qualitative and quantitative concerns of domain experts in a manner that not only guides their discussions and engineering activities, but can also be subjected to validation, simulation, and analysis. Additionally, the approach must classify the relationships between the different EEIS viewpoints and technical layers in order to address the design and operations concerns of the core management and technical stakeholders.

C. New Model-Based Capabilities for Reasoning and Analysis for EM work

Development of new model-based capabilities relies on the fact that views are merely projections of the central SysML model. The content of any view is determined by the set of concerns a stakeholder wishes to see (whether they are represented on a diagram, in a table, or as the output of a query or model analysis). Thus, our work on EM-1 and EM-2 EEIS aims to add structure and codification to the format of the core SysML model in order to facilitate more the more complex kind of views and queries described in the previous section.

This first involves standardizing the “information model,” or the way in which we capture attributes of elements like requirements, needlines, network routes, software interfaces, protocol stacks, etc. Building on a more rigorous information model, we then define relationships between these abstract concepts. This means asserting new rules: the conditions under which a physical connection may implement a logical one; which protocols may encapsulate and carry which data types and nest within other protocols; how to use those relationships to trace from a critical data exchange (a needline) down through the other logical and physical layers to collect the set of software interfaces, communications links, and even routing paths and hardware involved; and other rules necessary to address our more complex concerns. Figure 4 shows a conceptual example of such a set of relationships.

The next section will provide a more detailed examination of how we have chosen to represent the necessary structures and relationships.

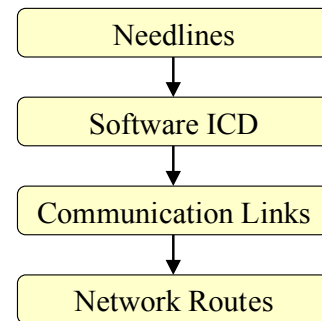


Figure 4: Mapping Between Logical Layers

II. MBSE Framework for EEIS Architecture Description

Both the development of the EEIS “information model” and its use in specification of the EM EEIS application problem were implemented via the Systems Modeling Language (SysML)⁶ plug-in in the MagicDraw⁷ modeling tool. MagicDraw is an institutionally chosen modeling tool that supports the SysML specification as the modeling notation for systems engineering. The information model is the combination of a lightweight profile, meta-modeling and model patterns. A profile is a set of elements that represent definitions of concepts. The profile elements, or stereotypes, may be connected to each other, representing relationships between concepts, and may contain sets of tags that become available on an element when the stereotype is applied. This section of the paper will discuss these more technical concepts that we are using to support addressing the stakeholders’ needs. We will provide a section for each previously discussed capability.

A. Configurations for Each Mission Phase - Specializing the EM-1 Mission

The first important construct we will discuss is the idea of configurations. By using separate model elements to represent each configuration (where the configuration represents one or more phases, linked together in the model), we strongly assert that all internal constructs are directly associated (through ownership) with that phase, as shown in Figure 5. This is a significant improvement over our previous method where we created one model element representing the entire mission (all phases) and the only way to tell what phase an element belongs to was through the name of the diagram on which it appears..

Figure 6 shows a simplified view of this composition

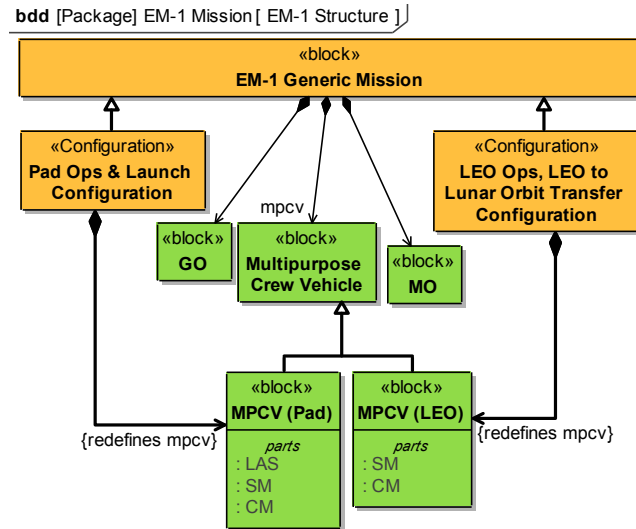


Figure 6: Example of EM-1 Composition Structure Showing Variation of the MPCV Between Phases

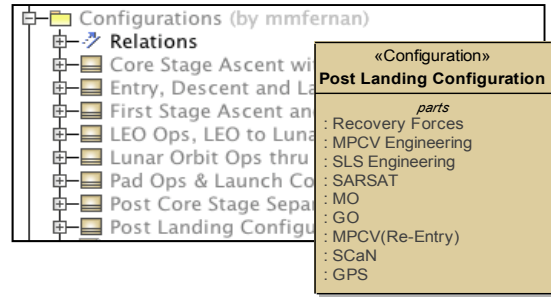


Figure 5: System Configurations as Distinct Model Elements

structure. The generic EM-1 mission is composed of many systems (for the sake of simplicity, not all EM systems are shown). The configurations of the EM-1 mission (On-pad and in Low Earth Orbit are shown) are ‘specializations’ of the generic EM-1 mission – they have the same set of systems. However, as we have shown in this figure, the MPCV configuration is different between On-pad and in LEO. We can capture these differences while retaining the common features of the MPCV by specializing MPCV and asserting by redefinition that the configurations shown ‘overwrite’ the inherited generic one.

This allows us to put common properties at the right level: for example, the generic MPCV contains features such as ports and parts which are still present in the configurations of the On-pad and in LEO phases. This structure ensures that model queries and analysis uses the correct configurations and keeps elements present only where they are supposed to exist.

B. Needlines as Instantiations of Mission Interface Requirements

The driving interface requirements for EM-1 and EFT-1, as chosen by NASA program management, are written without regard to mission phase or segment—they merely define the capabilities required by the systems that may be exercised during the mission. In a rigorous model, there is a separation between the assertion that two systems shall have the capability to exchange some data, and the assertion that they shall exchange the data at certain times under certain configurations and conditions. A needline is the assertion that a data exchange is required in a certain configuration/operational phase. This is accomplished in the model structure by first defining the pairwise data exchange (according to the EM-1 Interface Requirements Document, IRD), and then attaching it to the connections between model elements that it “governs.”

To be more technical: in SysML, a specification may be attached to a port that says what is allowed to flow through that port. These specifications may be attached to each other as seen in Figure 7, essentially making “pairs” of specifications. These pairs of flow definitions very neatly match the pairwise requirements that assert data exchange capabilities between systems.

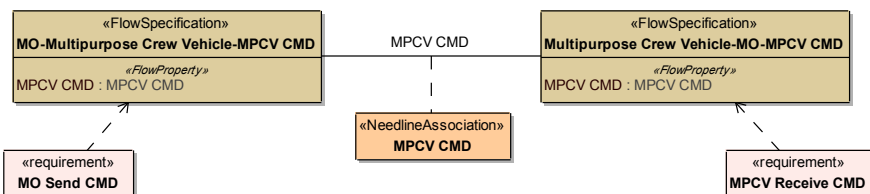


Figure 7: Interface Requirements linked to Paired Flow

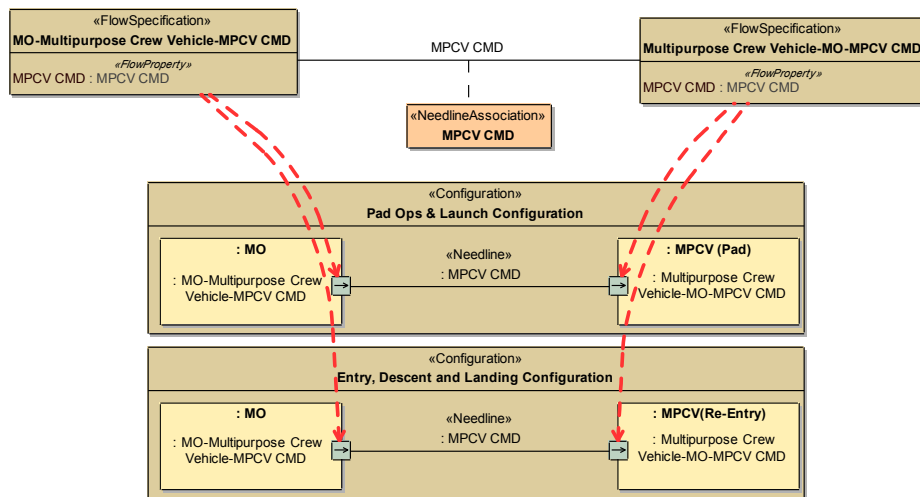


Figure 9: Applying the Paired Flow Specification Pattern To Configurations and Needlines

needline becomes an instance of the connection between the specifications. It is only allowed to become an instance of the generic connection if the specifications on the ports match - thereby ensuring that all needlines must map to a generic pair of specifications, thereby mapping to EM-1's paired requirements. The mapping of the flow specifications to the ports can be seen in the view as the name of the flow specification, prefaced with a colon (":") next to the port. Similarly, the instantiation of the link between the specifications shows up as the name of the link prefaced by a colon (":"). This is possible because the ports are owned by the definition of the system (not the instance owned by the configuration), which means they are available in all configurations.

Figure 8 illustrates another reason the separation of views into discrete configurations blocks is useful - because all connector elements are owned by the configuration block, the relationship between needline and configuration (and thus phase) is strongly asserted. Using the profile, we set the model software to infer the phase of each needline by looking at the phase of its owning block (configuration).

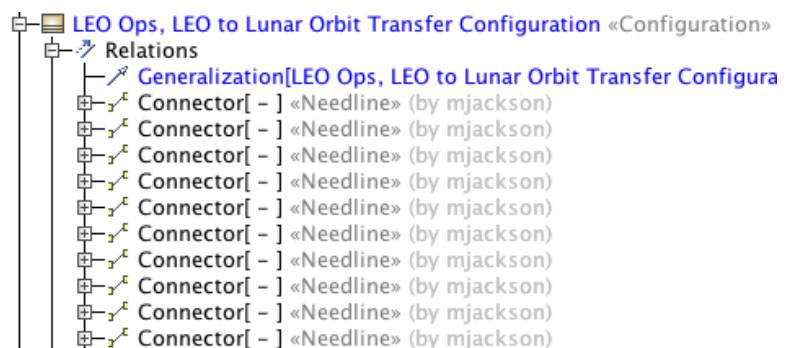


Figure 8: Configurations Own Their Needlines

C. Software Interfaces

Software Interfaces are constructed in the model in the same pattern as needlines—the capability to exchange data at the software level (i.e., specification of protocol stacks, etc.) exists independent of mission phase, but the exchange is only implemented in certain phases. Therefore we create paired specifications of flow, connect them, and instantiate them between systems when they are intended to be exercised. Thus we can reason about the implementation of software interfaces and the different paths that exist at any mission phase.

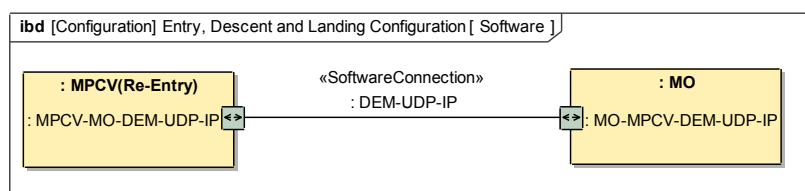


Figure 10: Software Connection Pattern

Software interfaces have different attributes of interest than needlines. For example, it is important to know more about the protocols and addressing implemented/required by the exchange, where the needline

tracks more qualitative and logical attributes. Where the specification of flow for needlines describes logical items (TLM, CMDs, etc.), the software interfaces do not care what the logical content of the data is, only that the exchange mechanisms are matched on both sides. For example, if the two systems have agreed to exchange Data Exchange Messages (DEMs) encapsulated in User Datagram Protocol (UDP) over Internet Protocol (IP), they do not care what the DEM is transporting as long as the protocols and addressing are in accordance with the interface specification. We will discuss the construction of protocol stacks a little later.

An important advantage is revealed already—as we mentioned, a DEM can transport various logical information items that are shown and whose exchange is described in the needline viewpoint. So far, it has been difficult and unwieldy to map the needline data exchanges to the software interfaces that are implementing them. By having the configuration own these well-understood structures, it is easy to query the model to find out what software interfaces are available in a given phase to transport the data in the needlines of the same phase. Before, it was not possible to do this, as there was no distinction between the connectivity of different mission phases.

D. Communications Viewpoint: Views and Constructs

Like needlines and software interfaces, the communications links are usages of a capability within a phase. The communications links carry protocols of interest, those below the IP layer. The communications links do not care

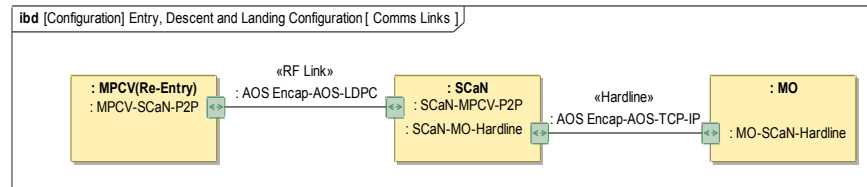


Figure 11: Communications Links - RF and Hardline

what kind of data or what logical data (and it could be many different pieces of logical data) flow between them, as long as they match the specifications asserted in the requirements.

In addition to the communications links, we also

want to track the existence of network interfaces. This view, often expressed within the communications viewpoint, establishes where some kind of point-to-point network connection exists or is required, although it may exist only logically (in reality, it might be carried over other networks or tunneled, utilize existing systems or involve a complex topology that is not relevant to specifying the connectivity at this level).

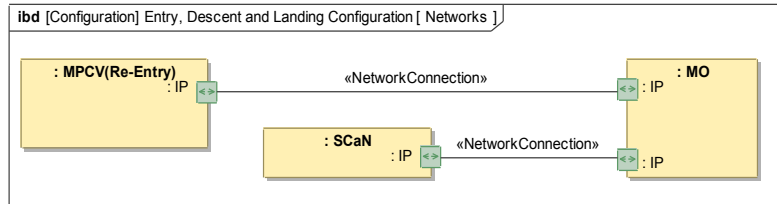


Figure 12: Network Connection Agreements

E. Routing/Hardware Constructs

Reasoning about the data paths at various levels of abstraction is a key capability to analyzing a complete EEIS Architecture. We have developed model constructs for representing the most concrete elements of the EEIS to address the concerns of the network engineer and to include them in a multi-layer system model.

We have constructs for routers, firewalls, addresses, workstations, subnets, etc., and paths between them. Strongly defining these constructs, in addition to mapping the software to the hardware and the hardware contained within our systems of interest, allows us to determine the set of hardware, networks, etc. involved within the implementation of a given needline or software interface.

This view tends to be more static, as it is ground-system focused, and generally the ground system configuration does not change along with the mission phase. As such, we collect ground system pieces into ‘invariants,’ which are static sets of hardware connectivity and rules that may be used in some or all of the mission phases. This also allows us to show a more facility-oriented view, and we map the facilities to the systems in order to allow reasoning about implementation of various data paths.

F. Protocol Stacks

As mentioned previously, we often want to know specifics of an exchange between two pieces of software or systems. At the needline layer, we are concerned with things like “commands” and “telemetry.” We can talk about

the format of video or a command at that level, but at the software interface level, we are concerned with how to move those items about - how to break them up and wrap up the pieces for transmission.

ibdd [DEM-UDP-IP]

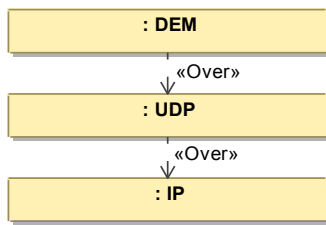


Figure 13: Protocol Stack

There are more rules asserted in protocol stack definitions - for example, each protocol element has relationships to other protocol elements and to datatype elements (CMD, TLM, etc.) asserting allowed encapsulation. Figure 14 shows the relationships between (some of the) protocols for EM-1, including support for tunneling, as IP stacks can be further encapsulated and sent over an intermediary IP network, for example. Another rule governs which interfaces carry what kinds of stacks. As EM-1 is using IP as a network interface layer, the software interface layer stops at IP. Thus, the software layer specifications allow protocol stack constructs to flow where their bottom layer is IP. <<Needline>> elements may carry datatypes (CMD, TLM, etc.), and communication links carry protocols whose bottom layers may be transmitted by RF, hardlines, or other networks.

Protocol Stack views analogous to those in EFT-1 may be created by inspecting the interfaces between the systems involved in implementing a particular data exchange. As we discuss in the next section, this involves cross-cutting analysis of the model across many levels of abstraction: not only do we need the high level data exchange, but the software interface that carries it, and the networks and communications links that carry the software interfaces. The stack can even be traced down to the hardware and routing paths and protocols, although we do not show an example of that in this paper.

A modeler may know the full stack for a needline outright, and implement that in the model—creating and assigning the appropriate software stack and the communication stack. If they do not know the applicable protocol stack, they may query it from the model, if all the communications links and software interfaces have been populated with their stacks. Thus, while the underlying structure of the model is different, the same sorts of protocol stack views may be created in EM-1 as for EFT-1 EEIS architecture modeling efforts.

G. Cross-Cutting Constructs

The power of the EM-1 modeling approach is the ability to observe, analyze, and query the intersection of different views representing the concerns of different engineering disciplines and domains. Simply creating the framework allowing population of the previous concerns allows us a huge amount of powerful reasoning about the model. On EM-1, we have built the underlying model structures necessary to consistently link and trace through the layers. By understanding the relationship between needlines, software interfaces, communications links, and hardware/network layers, we can easily query the model to understand critical paths, critical hardware, single points of failure, etc. We can either directly assert the mapping (with mapping relationships) between needlines and software interfaces, communications links and network routes, or we can ask the model to find routes for us, give us lists of involved hardware, compute link margins, etc. We can assess the effect on the system of changing one structure on the others: for example, compromising some subnets or software components can affect critical data exchanges, which we can identify by tracing through model layers and relationships.

We have created constructs for making protocol stack elements in the model, which can be exchanged like any other item across an interface. For example, Figure 13 shows a simple DEM-UDP-IPD stack. The <<Over>> relationship indicates (and bindingly asserts) the order of encapsulation in the stack. The stack element is a block that is exchanged by systems, across a connection - asserting an agreement by each system to conform to the specification of that stack. As seen previously in the Communications Constructs and Software Interfaces sections, the stacks are associated with the specification of flow from a software or communications port (analogous to the pattern used to define needlines).

bdd [Package] Protocols [Protocol Rules]

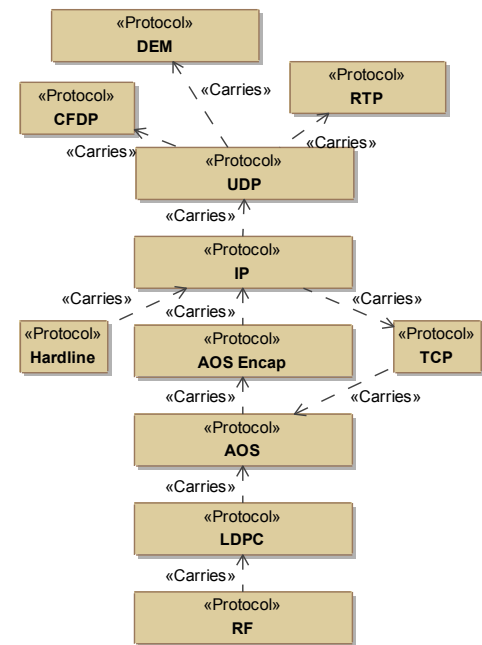


Figure 14: Subset of Allowed Relationships Between Protocols

An example of a cross-cutting view is shown in Figure 15. While this view would generally not be presented to a stakeholder, it is very useful in visualizing the traceability through the layers of abstraction. Here, we see how the agreement between two systems to exchange DEMs (themselves carrying CMDs, TLM, etc.) over UDP over IP

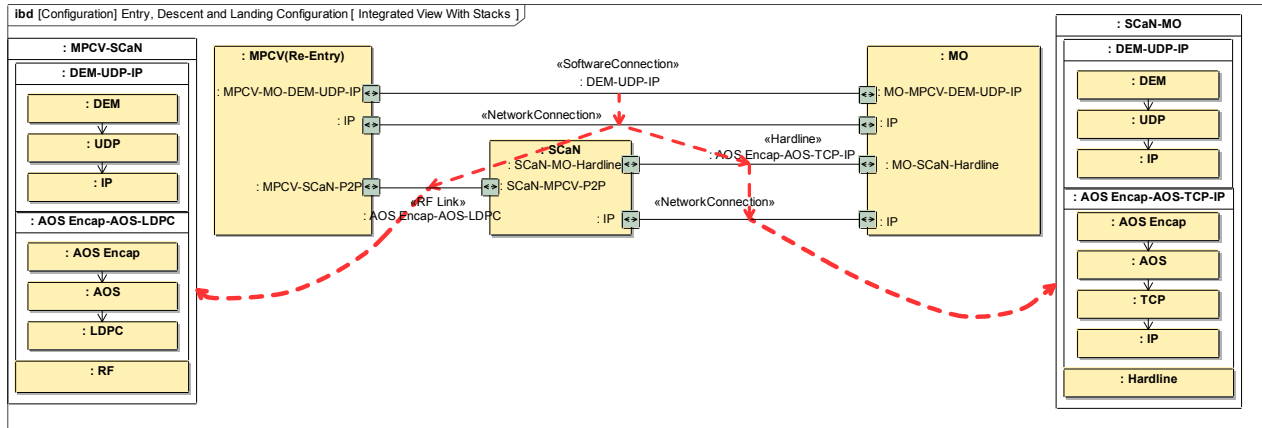


Figure 15: Example of a "Trace" Through the Levels of Abstraction

relies on the agreement that a logical network connection shall exist between the two. Further down, that network connection is actually made possible by the agreements those two systems have with a third system! And those agreements may be implemented a RF connection, or some other network that itself relies on a hardware/routing configuration (not shown) that may itself be very complex. Consider that there are far more than three systems exchanging many types of data across many interfaces in over eight mission phases, and the need for well understood rules, relationships, and reasoning becomes clear.

While we have thus far discussed only these four layers (and their requirements) across mission phases, we have the potential to do so much more. As we will discuss in the future work, we can incorporate functional models of the system and analyze the effect of hardware changes on EEIS functions. For example, we can also assess timing constraints, and include external models.

III. Modeling Lessons Learned

This more rigorous approach to constructing formal EEIS architecture description models has allowed for many new capabilities; however, constructing the ‘rules’ that define a system in the abstract is often challenging. The process involves analyzing the system to generalize the relationships between concepts, which is not an easy undertaking, as the concepts do not always fit neatly into complex model concerns. It requires a consistency in the existing system that is always achieved. Thus, there must be compromises, as a model does not allow qualitative exceptions, but reality may not always be placed neatly in a set of rules. Thus, the architect of the meta-model must study a wide variety of examples and concerns from the domains they seek to represent, in order to provide a place for all the concerns held by the domain experts and engineers. The domain experts and engineers must be able to articulate their concerns in abstract ways that can cover all possibilities, or recognize when something violates their own ideal rules and change it. The exercise of modeling a system is very valuable, as the places where the system does not fit nicely into the model reflects a place where either the system has violated some of its own rules, or the rules do not reflect reality. When these kinks are worked out, and everyone buys into the model, you have a product which everyone can stand behind and which may then be subjected to analysis and reasoning and the results may be regarded with a higher level of trust.

IV. Future Work

The constructs we have created have applications beyond just EM-1 and Human Space Flight. Currently, we plan to apply the mapping of hardware/network constructs, software interfaces, and data exchanges to diverse domains such JPL robotic missions and Smart Power Grids. We hope that as we continue to demonstrate value behind these constructs and views, they will be used more widely and tested even further. We can then more officially build suites of reasoning tools, understand and collect common analysis needs, and build infrastructure to open this area to engineers with reduced learning curve and rich user support to make intersectional modeling very simple.

Acknowledgments

The task was managed out of the Jet Propulsion Laboratory, a division of the California Institute of Technology, under contract with the National Aeronautics and Space Administration. The authors would like to thank Sanford Friedenthal for his feedback and guidance.

References

- ¹ Constellation Program Computing Systems Architecture Description Document, CxP 70078 Rev B, August 12, 2010. [ITAR controlled, not available in public domain].
- ² McVittie, T., Sindi, O., and Simpson, K., "Model-Based System Engineering of the Orion Flight Test 1 End-to-End Information System," 2012 IEEE Aerospace Conference, Big Sky, MT, 3-10 March 2012.
- ³ Simpson, K., Sindi, O., and McVittie, T., "Orion Flight Test 1 Architecture – Observed Benefits of a Model Based Engineering Approach," 2012 IEEE Aerospace Conference, Big Sky, MT, 3-10 March 2012.
- ⁴ Department of Defense Architecture Framework (DoDAF), U.S. Department of Defense, Version 2.02, August 2010.
- ⁵ Reference Architecture for Space Data Systems (RASDS) Recommended Practice, The Consultive Committee for Space Data Systems (CCSDS), Washington DC, USA, Issue 1, CCSDS 311.0-M-1, September 2008.
- ⁶ "OMG Systems Modeling Language," Object Management Group, Version 1.2, Needham, MA, June 2010.
- ⁷ MagicDraw, Software Package, Version 17, No Magic Inc., Allen, TX, 2011.